Vault Secret Management

Version: 1.0
Last Updated: 2025-11-26
Status: ✅ Story 1.4 Complete

📋 Table of Contents

🚀 Quick Start

Local Development (Dev Mode)

```
# 1. Start Vault
docker-compose up -d vault

# 2. Initialize secrets
docker-compose exec vault sh /vault/scripts/vault-init.sh

# 3. Verify secrets
docker-compose exec vault vault kv get secret/signature-router

# 4. Start application
./mvnw spring-boot:run

# 5. Verify health check
curl http://localhost:8080/actuator/health/vault
```

Expected Output:

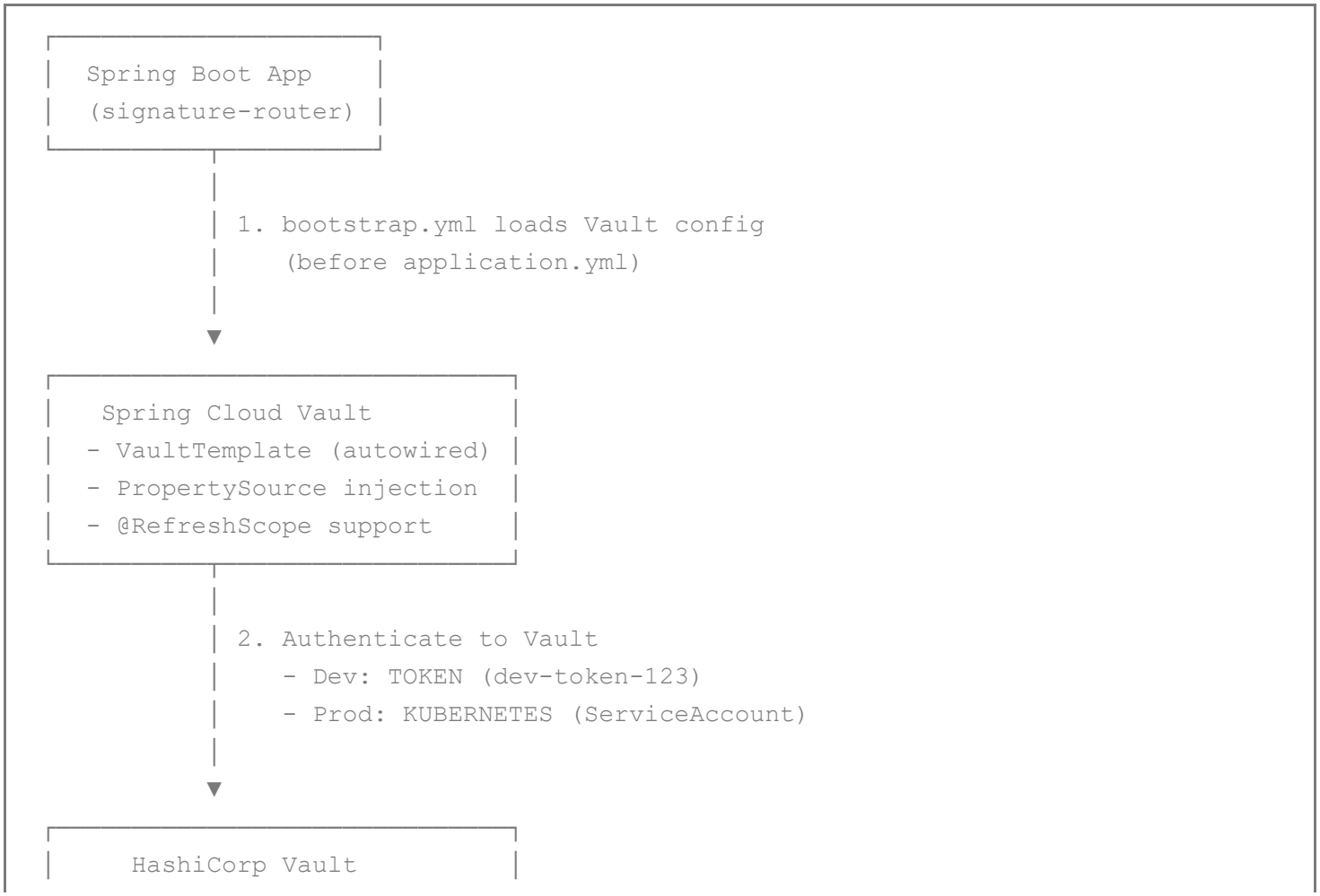
```
{
  "status": "UP",
  "details": {
    "vault": {
      "status": "UP",
      "details": {
        "version": "1.15.0"
      }
    }
  }
}
```


Accessing Vault UI

- **URL:** http://localhost:8200/ui
- **Token:** `dev-token-123`
- **Path:** `secret/signature-router`

🏗️ Architecture

Vault Integration Flow



```
|  - KV v2 Engine (versioned)  |
|  - Path: secret/signature-   |
|    router/                   |
|  - Secrets: db password,     |
|    API keys, licenses        |
└──────────────────────────────┘
```

Secret Loading Process

1. **Bootstrap Phase** (`bootstrap.yml` loads first):
   - Vault URI, authentication method, KV path configured
   - Spring Cloud Vault creates `VaultPropertySource`
2. **PropertySource Priority** (highest to lowest):
   - Vault secrets (`secret/signature-router/`)
   - `application-{profile}.yml`
   - `application.yml`
3. **Secret Injection**:
   - **@Value:** `@Value("${database.password}")` → resolved from Vault
   - **VaultTemplate:** Programmatic access via `VaultConfig.getSecret("key")`
4. **Dynamic Refresh** (optional):
   - Beans annotated with `@RefreshScope` reload when secrets change
   - Polling interval: 60s (dev), 300s (prod)

🔑 Secret Management

Managed Secrets

Secret Key	Description	Example Value (Dev)	Rotation Period
`database.password`	PostgreSQL password	`sigpass`	90 days (manual)
`kafka.sasl-jaas-config`	Kafka SASL auth config	`""` (placeholder)	N/A (prod only)
`twilio.api-key`	Twilio SMS API key	`test-twilio-key-123`	90 days

Secret Key	Description	Example Value (Dev)	Rotation Period
`twilio.api-secret`	Twilio SMS API secret	`test-twilio-secret-456`	90 days
`push-service.api-key`	Push notification API key	`test-push-key-789`	90 days
`biometric-sdk.license`	Biometric SDK license	`test-biometric-license`	365 days

Reading Secrets

Option 1: `@Value` Injection (Recommended)

```
@Component
public class DatabaseConfig {

    @Value("${database.password}")
    private String dbPassword;

    // Automatically injected from Vault
}
```

Option 2: VaultTemplate (Programmatic)

```
@Service
public class ProviderService {

    @Autowired
    private VaultConfig vaultConfig;

    public void sendSms() {
        String twilioApiKey = vaultConfig.getSecret("twilio.api-key");
        // Use twilioApiKey...
    }
}
```

Option 3: @RefreshScope (Dynamic Reload)

```
@Component
@RefreshScope  // Reloads when secrets change in Vault
public class TwilioService {

    @Value("${twilio.api-key}")
    private String apiKey;

    // apiKey refreshes every 60s (dev) if changed in Vault
}
```

Writing Secrets

⚠️ **WARNING:** Writing secrets programmatically is **NOT RECOMMENDED** in production. Secrets should be managed via CI/CD pipelines or Vault CLI.

```
// For testing/development ONLY
@Autowired
private VaultConfig vaultConfig;

vaultConfig.writeSecret("new-api-key", "value-123");
```

CLI Secret Management

```
# Read a secret
vault kv get secret/signature-router

# Update a specific secret
vault kv patch secret/signature-router database.password=newpass123

# List secret versions
vault kv metadata get secret/signature-router

# Rollback to previous version
vault kv rollback -version=2 secret/signature-router

# Delete secret (soft delete - can be undeleted)
vault kv delete secret/signature-router

# Permanently destroy secret
vault kv destroy -versions=3 secret/signature-router
```

🔄 Rotation Strategy

Current Approach (Manual Rotation)

1. **Update secret in Vault:**

```
vault kv patch secret/signature-router twilio.api-key=new-key-456
```

2. **Wait for auto-refresh** (if using `@RefreshScope`):
   - Dev: 60s polling interval
   - Prod: 300s polling interval
3. **OR restart application** (if not using `@RefreshScope`):

```
kubectl rollout restart deployment/signature-router
```

Future: Dynamic Secrets (Planned)

Goal: Vault generates short-lived credentials automatically.

Example: Database Dynamic Secrets

```
# Future implementation
spring:
  cloud:
    vault:
      database:
        enabled: true
        role: signature-router-role
        backend: postgres
```

Benefits:

- ✅ Credentials auto-rotated every 24h
- ✅ Reduces blast radius (compromised credentials expire quickly)
- ✅ Audit trail: Vault logs credential issuance

Timeline: Epic 5 (Security Hardening)

🛠️ Troubleshooting

Issue 1: Application Fails to Start (Vault Unavailable)

Symptom:

```
org.springframework.vault.VaultException: Cannot connect to Vault
```

Cause: Vault service not running or `fail-fast=true` enforced.

Solution:

```
# 1. Check if Vault is running
docker-compose ps vault

# 2. Start Vault
docker-compose up -d vault

# 3. Verify Vault health
docker-compose exec vault vault status

# 4. Restart application
./mvnw spring-boot:run
```

Issue 2: Secrets Not Found

Symptom:

```
IllegalStateException: Vault secret not found: secret/signature-router
```

Cause: Secrets not initialized in Vault.

Solution:

```
# Initialize secrets
docker-compose exec vault sh /vault/scripts/vault-init.sh

# Verify secrets exist
docker-compose exec vault vault kv get secret/signature-router
```

Issue 3: Token Expired (Production)

Symptom:

```
VaultException: 403 permission denied
```

Cause: Vault token expired (TTL exceeded).

Solution:

For **Kubernetes authentication** (production):

- Spring Cloud Vault **auto-renews** tokens before expiry
- If renewal fails, restart pod:

```
kubectl rollout restart deployment/signature-router
```

For **TOKEN authentication** (dev):

- Token `dev-token-123` never expires in dev mode
- In prod, NEVER use TOKEN auth (use KUBERNETES auth)

Issue 4: Secrets Not Refreshing

Symptom: Updated secret in Vault, but application still uses old value.

Cause: Bean not annotated with `@RefreshScope`.

Solution:

1. **Add `@RefreshScope` annotation:**

```
@Component
@RefreshScope  // <-- Add this
public class MyService {
    @Value("${twilio.api-key}")
    private String apiKey;
}
```

2. **OR restart application:**

```
./mvnw spring-boot:run
```

Issue 5: Vault Sealed

Symptom:

```
Vault is sealed
```

Cause: Vault requires manual unseal (not in dev mode).

Solution:

Dev Mode (auto-unsealed):

- Vault auto-unseals in dev mode (no action needed)

Production (manual unseal required):

```
# Unseal Vault (requires 3 unseal keys)
vault operator unseal <key1>
vault operator unseal <key2>
vault operator unseal <key3>
```

🏭 Production Considerations

Authentication: KUBERNETES vs TOKEN

Auth Method	Use Case	Token Source	Renewal
TOKEN	Local dev ONLY	Hardcoded `dev-token-123`	Never expires (dev mode)
KUBERNETES	UAT, Prod	ServiceAccount JWT	Auto-renewed by Spring Cloud Vault

Production Configuration:

```
# bootstrap-prod.yml
spring:
  cloud:
    vault:
      uri: https://vault-prod.internal:8200
      authentication: KUBERNETES
      kubernetes:
        role: signature-router-prod
        service-account-token-file:
/var/run/secrets/kubernetes.io/serviceaccount/token
      ssl:
        trust-store: classpath:truststore.jks
        trust-store-password: ${TRUSTSTORE_PASSWORD}
```

High Availability (HA)

Vault Cluster Setup:

- ✅ **3+ Vault nodes** (Raft consensus)
- ✅ **Load balancer** fronting Vault nodes (`vault-prod.internal:8200`)
- ✅ **Auto-unseal** via AWS KMS / Azure Key Vault (no manual unseal keys)
- ✅ **Integrated storage** (Raft) or **Consul** backend

Spring Boot Configuration:

```
# No changes needed - Vault HA transparent to clients
spring:
  cloud:
    vault:
      uri: https://vault-prod.internal:8200  # Load balancer URL
```

Security Best Practices

1. **NEVER use dev mode in production**
   - ⚠️ Dev mode: in-memory storage, auto-unsealed, root token
   - ✅ Prod mode: persistent storage, manual unseal, Kubernetes auth
2. **Enable TLS for Vault**

   ```
   spring:
     cloud:
       vault:
         uri: https://vault-prod.internal:8200  # HTTPS, not HTTP
         ssl:
           trust-store: classpath:truststore.jks
   ```

3. **Use Kubernetes authentication (not TOKEN)**
   - ✅ Tokens auto-rotated via ServiceAccount
   - ✅ No hardcoded credentials in configs
4. **Enable fail-fast mode**

   ```
   spring:
     cloud:
       vault:
         fail-fast: true  # App won't start if Vault down
   ```

5. **Audit logging**

- Vault audit logs: who accessed which secrets, when
- Enable in production:

```
vault audit enable file file_path=/vault/logs/audit.log
```

6. **Least privilege policies**
   - Vault policies grant minimal permissions
   - Example policy:

```
path "secret/data/signature-router" {
  capabilities = ["read"]
}
```

Monitoring & Observability

Metrics to Monitor:

1. **Vault Health Check:**

```
curl http://localhost:8080/actuator/health/vault
```

2. **Token TTL:**
   - Monitor `vault.token.ttl` metric
   - Alert if TTL < 1 hour (renewal might fail)
3. **Vault Cluster Health:**

```
vault status  # Check sealed status, HA leader
```

4. **Secret Access Audit:**
   - Parse Vault audit logs
   - Alert on unauthorized access attempts

Prometheus Metrics:

```
# Exposed via Spring Boot Actuator
management:
  metrics:
    export:
      prometheus:
        enabled: true
```

📚 Additional Resources

- Vault Official Documentation
- Spring Cloud Vault Reference
- Vault Best Practices
- **Architecture Documentation** - PseudonymizationService, secret management

---

Last Updated: 2025-11-26 (Story 1.4)
Next Steps: Epic 5 (Security Hardening) - Implement Vault dynamic secrets for auto-rotation